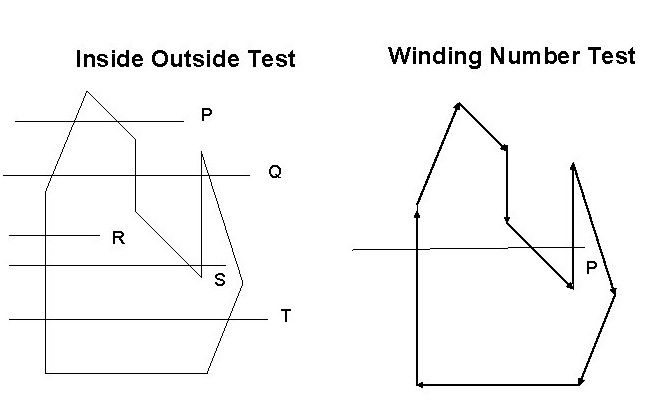
INSIDE TESTS

In Computer Graphics, Inside Outside is performed to test whether a given point lies inside of a closed polygon or not. Mainly, there are two methods to determine a point is interior/exterior to polygon:

1. 1. Even-Odd / Odd-Even Rule or Odd Parity Rule
2. 2. Winding Number Method

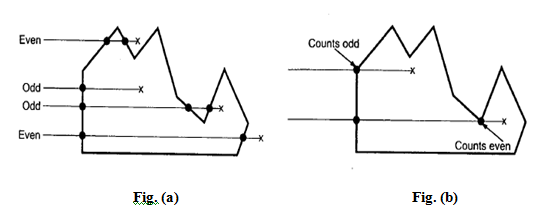


**Even-Odd Rule / Odd Parity Rule**

It is also known as crossing number and ray casting algorithm. The algorithm follows a basic observation that if a ray coming from infinity crosses through border of polygon, then it goes from outside to inside and outside to inside alternately. For every two crossings, point lies outside of polygon.

**Algorithm:**

1. Construct a line segment from point to be examined to point outside of a polygon.
2. Count the number of intersections of line segment with polygon boundaries.
3. If Odd number of intersection, then Point lies inside of Polygon.
4. Else, Point lies outside of polygon.



This test fails in case line segment intersects at vertex point. To handle it, few modifications are made. Look at other end points of two line segments of polygon.

* If end points lie is at same side of constructed line segment, then even number of intersection is considered for that intersection point.
* If end points lie at opposite side of it, then odd number of intersection is considered.

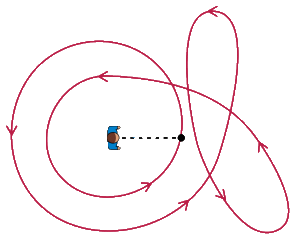
**Example**  
In Fig.(a) , it can be seen that points having total odd number of intersections lies inside the polygon and vice-versa. In Fig.(b) representing special case, odd/even count is taken according to side orientation.

**Winding Number / Non-Zero Algorithm**

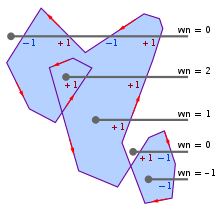
Alternative algorithm to perform test is Winding Number algorithm. A winding Number is calculated for given point with respect to polygon. If winding number is non-zero, then point lies inside the polygon. Else, it lies outside of polygon.

**Calculation of Winding Number**

Conceptually,to check a point P, construct a line segment starting from P to point on boundary.Treat line segment to be elastic pinned at P. Stretch other end of elastic around the polygon for one complete cycle. Check how many times elastic has been wounded around point P. If count is non-zero, then point lies inside of polygon. Else, outside of polygon.



Another way to score up winding number is to assign a score for each intersection with boundary of polygon and sum these numbers. The score is given by considering direction of edge of polygon with respect to line segment constructed. Hence, directions are assigned to each edge of polygon in counter-clock manner. If side of edge starts from below of constructed line, then score -1 is given. If edge starts from above of constructed line then score 1 is given. Else, score 0 is given.



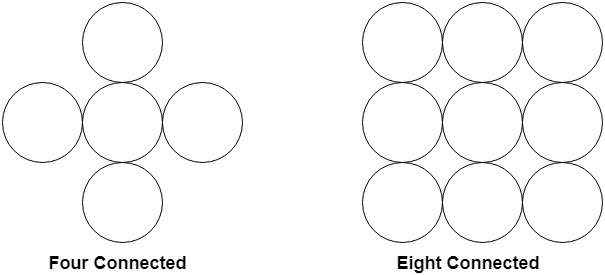
**Example**  
For a given figure,  
**1.** For a top-most point, Winding number = (-1) + (1) + (-1) + (1) = 0 , lies outside.  
**2.** For bottom-most point, Winding number = -1 , lies inside.

Filled Area Primitives:

Region filling is the process of filling image or region. Filling can be of boundary or interior region as shown in fig. Boundary Fill algorithms are used to fill the boundary and flood-fill algorithm are used to fill the interior.

Boundary Filled Algorithm:

This algorithm uses the recursive method. First of all, a starting pixel called as the seed is considered. The algorithm checks boundary pixel or adjacent pixels are colored or not. If the adjacent pixel is already filled or colored then leave it, otherwise fill it. The filling is done using four connected or eight connected approach.



Four connected approaches is more suitable than the eight connected approaches.

**1. Four connected approaches:** In this approach, left, right, above, below pixels are tested.

**2. Eight connected approaches:** In this approach, left, right, above, below and four diagonals are selected.

Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.

Problem with recursive boundary fill algorithm:

It may not fill regions sometimes correctly when some interior pixel is already filled with color. The algorithm will check this boundary pixel for filling and will found already filled so recursive process will terminate. This may vary because of another interior pixel unfilled.

So check all pixels color before applying the algorithm.

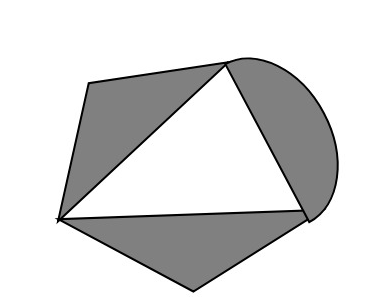
Algorithm:

1. Procedure fill (x, y, color, color1: integer)
2. **int** c;
3. c=getpixel (x, y);
4. **if** (c!=color) (c!=color1)
5. {
6. setpixel (x, y, color)
7. fill (x+1, y, color, color 1);
8. fill (x-1, y, color, color 1);
9. fill (x, y+1, color, color 1);
10. fill (x, y-1, color, color 1);
11. }

# Flood Fill Algorithm:

In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.



In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

## Disadvantage:

1. Very slow algorithm
2. May be fail for large polygons
3. Initial pixel required more knowledge about surrounding pixels.

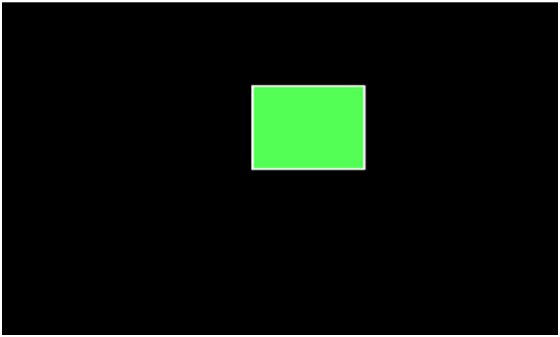
## Algorithm:

1. Procedure floodfill (x, y,fill\_ color, old\_color: integer)
2. If (getpixel (x, y)=old\_color)
3. {
4. setpixel (x, y, fill\_color);
5. fill (x+1, y, fill\_color, old\_color);
6. fill (x-1, y, fill\_color, old\_color);
7. fill (x, y+1, fill\_color, old\_color);
8. fill (x, y-1, fill\_color, old\_color);
9. }
10. }

### Program1: To implement 4-connected flood fill algorithm:

1. #include<stdio.h>
2. #include<conio.h>
3. #include<graphics.h>
4. #include<dos.h>
5. **void** flood(**int**,**int**,**int**,**int**);
6. **void** main()
7. {
8. intgd=DETECT,gm;
9. initgraph(&gd,&gm,"C:/TURBOC3/bgi");
10. rectangle(50,50,250,250);
11. flood(55,55,10,0);
12. getch();
13. }
14. **void** flood(intx,inty,intfillColor, intdefaultColor)
15. {
16. **if**(getpixel(x,y)==defaultColor)
17. {
18. delay(1);
19. putpixel(x,y,fillColor);
20. flood(x+1,y,fillColor,defaultColor);
21. flood(x-1,y,fillColor,defaultColor);
22. flood(x,y+1,fillColor,defaultColor);
23. flood(x,y-1,fillColor,defaultColor);
24. }
25. }

**Output:**



### Program2: To implement 8-connected flood fill algorithm:

1. #include<stdio.h>
2. #include<graphics.h>
3. #include<dos.h>
4. #include<conio.h>
5. **void** floodfill(intx,inty,intold,intnewcol)
6. {
7. **int** current;
8. current=getpixel(x,y);
9. **if**(current==old)
10. {
11. delay(5);
12. putpixel(x,y,newcol);
13. floodfill(x+1,y,old,newcol);
14. floodfill(x-1,y,old,newcol);
15. floodfill(x,y+1,old,newcol);
16. floodfill(x,y-1,old,newcol);
17. floodfill(x+1,y+1,old,newcol);
18. floodfill(x-1,y+1,old,newcol);
19. floodfill(x+1,y-1,old,newcol);
20. floodfill(x-1,y-1,old,newcol);
21. }
22. }
23. **void** main()
24. {
25. intgd=DETECT,gm;
26. initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
27. rectangle(50,50,150,150);
28. floodfill(70,70,0,15);
29. getch();
30. closegraph();
31. }

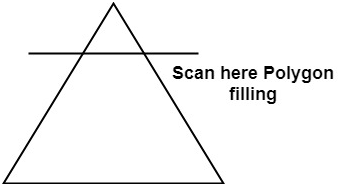
**Output:**



Scan Line Polygon Fill Algorithm:

This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement. The polygon is filled with various colors by coloring various pixels.

In above figure polygon and a line cutting polygon in shown. First of all, scanning is done. Scanning is done using raster scanning concept on display device. The beam starts scanning from the top left corner of the screen and goes toward the bottom right corner as the endpoint. The algorithms find points of intersection of the line with polygon while moving from left to right and top to bottom. The various points of intersection are stored in the frame buffer. The intensities of such points is keep high. Concept of coherence property is used. According to this property if a pixel is inside the polygon, then its next pixel will be inside the polygon.



Side effects of Scan Conversion

**1. Staircase or Jagged:** Staircase like appearance is seen while the scan was converting line or circle.

**2. Unequal Intensity:** It deals with unequal appearance of the brightness of different lines. An inclined line appears less bright as compared to the horizontal and vertical line.

